

(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号

特開平6-348595

(43) 公開日 平成6年(1994)12月22日

(51) Int.Cl.⁵
G 0 6 F 12/08

識別記号 庁内整理番号
3 1 0 Z 7608-5B

F I

技術表示箇所

審査請求 未請求 請求項の数5 F D (全 9 頁)

(21) 出願番号 特願平5-163148

(22) 出願日 平成5年(1993)6月7日

(71) 出願人 000005108

株式会社日立製作所

東京都千代田区神田駿河台四丁目6番地

(72) 発明者 久島 伊知郎

神奈川県川崎市麻生区王禅寺1099番地 株
式会社日立製作所システム開発研究所内

(72) 発明者 海永 正博

神奈川県川崎市麻生区王禅寺1099番地 株
式会社日立製作所システム開発研究所内

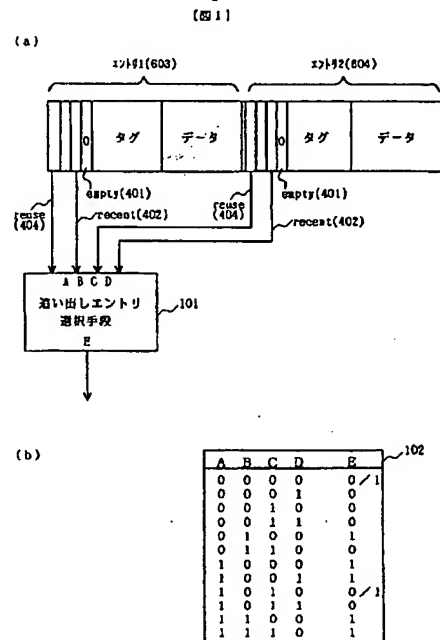
(74) 代理人 弁理士 笹岡 茂 (外1名)

(54) 【発明の名称】 キャッシュ装置

(57) 【要約】

【目的】 特定のデータを優先的にキャッシュに残しておくことを可能とするキャッシュ装置を提供する。

【構成】 キャッシュの各エントリにデータの追い出しを優先的に阻止する reuse ビットを設け、図1の場合、2セットのエントリのうち一方だけの reuse ビット (404) が真になっている場合は、他方のエントリを追い出しエントリとして選択する。両セットの reuse ビットとも真、または両セットとも偽になっている場合は、recent ビット402が偽になっているエントリを選択する。ロード命令の命令語には REUSE 指定があるかどうかを示すビットを設け、このビットが真であれば、当該ロード命令でアクセスしたデータを含むキャッシュエントリの reuse ビット (404) は真となる。



【特許請求の範囲】

【請求項1】 中央処理装置に接続され、それぞれがデータ領域および制御情報領域を有する複数のエントリを備えるキャッシュ装置において、エントリ中のデータのキャッシュからの追い出しを優先的に阻止するreuse情報を前記制御情報の1つとして設け、

キャッシュ置換時に前記制御情報に基づきキャッシュから追い出すエントリを選択しかつ前記reuse情報の設定されていないエントリを優先して選択する選択手段 10 を具備したことを特徴とするキャッシュ装置。

【請求項2】 請求項1記載のキャッシュ装置において、前記中央処理装置におけるメモリデータの参照と前記reuse情報の値の設定をする命令の実行に応じ、前記エントリにデータを格納すると共に前記reuse情報の値を設定する手段を具備したことを特徴とするキャッシュ装置。

【請求項3】 請求項1または請求項2記載のキャッシュ装置において、前記キャッシュ装置をセットアソシアティブ方式としたことを特徴とするキャッシュ装置。 20

【請求項4】 請求項1または請求項2記載のキャッシュ装置において、前記キャッシュ装置を2ウェイ・セットアソシアティブ方式としたことを特徴とするキャッシュ装置。

【請求項5】 請求項1または請求項2記載のキャッシュ装置において、前記reuse情報を1ビット情報としたことを特徴とするキャッシュ装置。

【発明の詳細な説明】

【0001】

【産業上の利用分野】 本発明はキャッシュ装置に係り、特に重複使用されるデータを優先的にキャッシュに残すのに好適なキャッシュ装置に関する。 30

【0002】

【従来の技術】 キャッシュは、高速に動作するCPUとアクセス速度の遅い主メモリ（以降単にメモリと呼ぶ）の速度のギャップを埋めるために設けられたアクセス速度の速い小容量の記憶装置である。CPU、キャッシュとメモリの関係を簡単に示すと図2のようになる。プログラムを実行する際、CPUが参照するメモリ内容は、もしキャッシュ上に存在していればそれを使い、なければメモリまでアクセスしに行く。キャッシュはCPUに非常に近く、しかも高速なので、CPUが必要とする多くのデータがキャッシュ上にあれば、非常に高速な処理が期待できる。キャッシュが有効に働くには、アクセス頻度の高いアドレスのデータをキャッシュに置いておくことが必要である。一般的なプログラムのメモリアクセスのパターンを解析すると、次のような特徴が現れる場合が非常に多い。

1. あるのアドレスのデータに対するアクセスは比較的短い時間内に再発する。

2. ある一定時間内にアクセスされるデータは比較的近いアドレスに分布する。

前者は「時間的局所性」、後者は「空間的局所性」と呼ばれる。

【0003】 前者の時間的局所性が表すところは「一度アクセスされたアドレスは近い将来再びアクセスされる」ということである。すなわち、今アクセスされたデータをキャッシュにとっておけば、次回そのアドレスが再び参照されたときにはそのデータはキャッシュにあるためアクセス時間が短くて済む。したがって、通常のキャッシュでは、CPUが参照するデータがキャッシュ上にない場合はそれを必ずキャッシュにもってきて、将来の再参照に備えるようにしている。一方、キャッシュは後者の空間的局所性も利用している。空間的局所性の意味するところは「あるアドレスがアクセスされたら、近い将来その近くのアドレスもアクセスされる」ということである。すなわち、データを新たにメモリからキャッシュに持ってくるときは、今アクセスされたアドレスのデータだけをキャッシュに持ってくるのではなく、その付近のデータも一緒にキャッシュに持ってくるようにすれば、将来その付近のアドレスのデータへのアクセスがあってもそのデータはキャッシュにあるためアクセス時間が短くて済むことが多い。このため、キャッシュは一定長（十数バイトから百バイト程度）のラインごとに分割されており、キャッシュとメモリとのデータのやりとりはライン単位に行われている。例えば、ある1つのデータを読むためにはそれを含む1ラインのデータ全てをキャッシュに持ってくるようにしている。

【0004】 キャッシュはメモリアクセス時間を短縮するのに有効であるが、局所性がないプログラムに対しては無駄になることがある。例えば次のプログラムを考えてみる。なお、floatは浮動小数点データの宣言を意味し、i++はiに+1することを意味する。

```
float a[1000];
```

```
for (i=0; i<100; i++) {
```

```
    ...a[i*10] ...
```

```
}
```

ここで配列aへの参照を考える。このプログラムで参照される配列aの要素は、a[0]、a[10]、a[20]、…、a[990]である。それぞれ必ず違った要素が参照されるので、時間的局所性は存在しない。また空間的局所性を考えると、例えばキャッシュの1ラインが32バイト、配列要素1個が4バイトという想定のもとでは、1ラインには配列の要素は8個分しか保持されず、上のプログラムのように10個おきにデータがアクセスされる場合にはライン中の他のデータは全く利用されない。すなわち（空間の大きさを32バイトとすれば）空間的局所性もない。

【0005】 このようなキャッシュへの無駄なロードを避けるため、キャッシュバイパスという機能（命令）を 50

もつプロセッサがある。これはメモリ中のデータを参照する命令について、そのデータがキャッシュ中に存在しない場合でも、それをキャッシュに置かないで直接使用することを指示できるようにしたものである。上のプログラムの配列aへの参照のようにキャッシュにデータを置いて無駄なことがわかっている場合は、その命令を使い、aの要素をキャッシュに置かないようにすることができる。このような機能を備えたプロセッサについては、例えば、リー：「オン・ザ・フローティング・ポイント・パフォーマンス・オブ・ザ・i860・マイクロプロセッサ」、インターナショナル・ジャーナル・オブ・ハイ・スピード・コンピューティング、ヴォリューム4、ナンバー4（1992）、第251頁から267頁（K. Lee: "On the Floating

```
float a[100], b[1000][100];
for (i=0; i<1000; i++) {
    for (j=0; j<100; j++) {
        ...a[j]...
        ...b[i][j]...
    }
}
```

このプログラムは2重ループで、aとbの2つの配列を参照している。aは1次元配列で大きさは100*4=400バイト、bは2次元配列で大きさは1000*100*4=400キロバイトである。キャッシュのサイズが32キロバイトという想定であれば、これらの全てのデータをキャッシュに収めることは不可能である。配列aは内側のループではa[0], a[1], a[2], ..., a[99]と100個の要素が参照される。外側のループによりこれが1000回繰り返され、同じ要素が1000回参照されることになる。すなわち配列aの参照は時間的局所性と空間的局所性（連続するアドレスを順次アクセスするので）を併せ持つ可能性が高い。配列aの大きさはキャッシュサイズより小さいので、全体がキャッシュに収まる。したがって1度配列要素をキャッシュに持ってくれば残りの999回の参照は、理想的にはキャッシュヒット（データがキャッシュ上に存在）するはずである。一方、配列bはループ全体で順にb[0][0], b[0][1], ..., b[0][99], b[1][0], b[1][1], ..., b[1][99], b[2][0], b[2][1], ..., b[999][99]と1000*100=100000個の要素が参照される。同じ要素が2度参照されることはないので時間的局所性はないが、空間的局所性はあるのでキャッシュが有効に働く。すなわち、キャッシュの1ラインに配列要素が8個分はいるという想定では、1つの要素（例えばb[0][0]）を参照すると後続する7つの要素（b[0][1], ..., b[0][7]）も一緒にキャッシュにはいるので、後の7回の参照はキャッシュヒットになる。

*Point Performance of the i860 Microprocessor", International Journal of High Speed Computing, Vol. 4, No. 4 (1992), pp. 251-267). に記載がある。また一般にキャッシュ技術に対する解説としては、情報処理、ボリューム33、ナンバー11（1992）、第1348頁から1357頁に記載がある。

【0006】

【発明が解決しようとする課題】上記従来技術では、空間的局所性と時間的局所性を持つデータを扱うプログラムでキャッシュが有効利用できない場合がある。これを以下のプログラムで説明する。

【0007】さて、配列bの大きさはキャッシュサイズよりも大きいので、その全体をキャッシュにいれることはできない。とするといつかは、以前キャッシュに入っていたデータを追い出して新たにそこにデータを入れるということが必要になる。このとき、どのデータを追い出すかというのが問題になるが、上のプログラムの場合、配列aのデータを追い出すよりも、配列bのデータを追い出したほうがよい。なぜなら配列aの各要素は何度も参照されるけれども配列bのデータは1度しか参照されないからである。一般のキャッシュでは、どのデータをキャッシュから追い出すかはランダムまたはLRU（Least Recently Used）という選択方法を採用している。ランダムは追い出すエントリを文字通りランダムに選ぶもので、データの使用状況は全く考慮されない。LRUは最後にアクセスされた時刻が最も古いデータをキャッシュから追い出すという選択方法である。これは、言ってみればデータの過去の参照状況をみてそれをキャッシュに残すかどうか追い出すかを決めるものである。いずれにしても、そのデータが将来再使用されるかどうかを直接考慮したものではない。従って上のプログラムの場合、配列bが追い出されて配列aが残されるという保証はなかった。すなわち従来技術では、特定のデータ（例えば将来再使用されることが明らかデータ）をキャッシュから追い出さないようにする、ということができなかった。また、プログラムまたはコンパイラが、特定のデータに対し、そのデータが再使用される可能性が高いのでそれをキャッシュから追い出さないようにするという指示を与えることができなかった。この結果、プログラムによってはキャッシュが有効

利用されない(キャッシュヒットが少ない)場合があるという問題点があった。本発明の目的は、特定のデータをキャッシュからなるべく追い出さないようにすることが可能なキャッシュ装置を提供することにある。本発明の別の目的は、プログラムまたはコンパイラが、特定のデータに対し、そのデータをなるべくキャッシュから追い出さないようにするという指示を与えることができる、キャッシュ装置を含む情報処理装置を提供することにある。

【0008】

【課題を解決するための手段】上記目的を達成するため、中央処理装置に接続され、それぞれがデータ領域および制御情報領域を有する複数のエントリを備えるキャッシュ装置において、エントリ中のデータのキャッシュからの追い出しを優先的に阻止するreuse情報を前記制御情報の1つとして設け、キャッシュ置換時に制御情報に基づきキャッシュから追い出すエントリを選択し、かつreuse情報の設定されていないエントリを優先して選択する選択手段を設けている。また、中央処理装置におけるメモリデータの参照とreuse情報の値の設定をする命令の実行に応じて、エントリにデータを格納すると共にreuse情報の値を設定する手段を設けている。さらに、キャッシュ装置をセットアソシアティブ方式のキャッシュ装置としている。さらにまた、キャッシュ装置を2ウェイ・セットアソシアティブ方式のキャッシュ装置としている。さらにまた、前記reuse情報を1ビット情報としている。

【0009】

【作用】課題で挙げたプログラムを例にとって作用の説明をする。プログラムはループの中でaとbの2つの配列要素の参照があるが、そのうちaの要素(データ)を参照する(ロードする)命令では命令語で、該データをなるべくキャッシュから追い出さないように指示し、bの要素を参照する命令では、そのような指示をつけない。これはプログラムまたはコンパイラより行なわれる。メモリアクセス命令は命令語中にその情報を指定するビットを持つようにしているので、これが可能である。データをメモリからキャッシュにフェッチする際、命令語に上記指定があれば、フェッチしたデータを格納するエントリに、当データはなるべくキャッシュから追い出されないようにする、という情報(reuse情報)を付け加える。よってaのデータを格納するキャッシュエントリにはそのような情報が設定され、bの要素を格納するエントリには設定されない。以前或るキャッシュエントリに入っていたデータを追い出して新たにそこに別のデータをいれるということが必要になった場合、上記情報が設定されていないエントリが優先的にキャッシュから追い出される。例の場合、aのデータを格納したエントリよりもbのデータを格納したエントリからデータが追い出される。このように、特定のデータを

キャッシュからなるべく追い出さないようにすることが可能なキャッシュ装置が得られる。また、プログラムまたはコンパイラが、特定のデータに対し、そのデータをなるべくキャッシュから追い出さないようにするという指示を与えることができる、キャッシュ装置を含む情報処理装置が得られる。

【0010】

【実施例】以下本発明の一実施例を説明する。図2は本発明のキャッシュを含む情報処理装置の構成図である。キャッシュ202はアクセス速度の速い小容量の記憶装置で、主メモリ204(以降単にメモリ)はアクセス速度の遅い大容量の記憶装置である。キャッシュとメモリはともにバス203とつながっており、またCPU201はキャッシュを介してバスとつながっている。プログラムを実行する際、CPUが参照するメモリ内容は、もしキャッシュ上に存在していればそれを使い、なければメモリまでアクセスしていく。なお、図2の構成図は従来のキャッシュでも同じである。

【0011】図3は本発明の一実施例のキャッシュの構成図である。本キャッシュは2ウェイセットアソシアティブ(セット1(301)とセット2(302)の2つのセットからなるアソシアティブキャッシュ)である。キャッシュ容量は全体で8kバイト、すなわち1つのセットの容量は4kバイトであり、ここにデータがラインを単位に配列的に格納されている。1セットは0番から127番までの128個のライン(エントリ)306からなる。このラインの番号をインデクス307と呼ぶ。1ラインには32バイトのメモリデータが格納される(32*128=4096バイト)。1語が4バイトであるとする、1つのエントリには8語格納可能である。1つのラインは制御ビット303、タグ304、データ305の3つの部分からなる。これについては図4で説明する。

【0012】図4は本発明のキャッシュの1ラインの構成を示した図である。ライン(306)は制御情報が格納される制御情報領域とデータが格納されるデータ領域を有し、制御情報としてempty情報、recent情報、dirty情報、およびreuse情報が設けられ、実施例では各情報は1ビット情報とされ、制御情報領域は制御ビット(303)からなる。また、データ領域は、実施例ではタグ(304)、データ(305)からなる。そして制御ビットはemptyビット(401)、recentビット(402)、dirtyビット(403)、およびreuseビット(404)からなる。データはメモリ中のデータの値を格納するフィールドである。タグは今格納しているデータが、どのメモリアドレスに対応するものかを特定するフィールドである。emptyビットは当ラインが空きであるかを示す(真のとき空き)。recentビットは当セットのラインが、もう一方のセットのラインよりも後にアクセス

されたかどうか、すなわち最新のアクセスであるかを示す(真のとき最新)。recentビットは、本キャッシュがLRU(Least Recently Used)に基づく置換方法を用いているため必要となる。dirtyビットは当キャッシュラインに書き込みがあったかを示す(真のとき書き込みあり)。reuseビットは、当ラインのデータが将来再び使用される可能性が高いことを示す。このビットは当該メモリアドレスを参照する命令により設定され、キャッシュ置き換えが必要になったときに参照されるが、これについては後で詳しく説明する。

【0013】図5はメモリアドレスを2進表現したときの構成を示す図である。アドレスは32ビットで、タグ501、インデクス502、ポジション503の3つの部分にわかれる。タグは20ビットでこれがキャッシュラインのタグ(304)と対応する。インデクスは7ビットでこれをキャッシュラインのインデクスとして用いる(2の7乗=128であるからこの7ビットによりキャッシュラインが特定できる)。ポジションは5ビットで、ライン中のバイト位置を特定するのに使われる(2の5乗=32であるからこの5ビットでバイト位置が特定できる)。

【0014】次に図6を用いて、メモリアクセスが生じたときの本発明のキャッシュの振舞いを説明する。さて、メモリアクセスが発生したとする。この時メモリアドレス(602)のうちインデクス(502)の値をインデクスとしてキャッシュライン配列のエントリを特定する。2ウェイのセットアソシアティブであるので、1つのインデクスに対しエントリは2つ(セット1のエントリ603とセット2のエントリ604)特定される。次に当該のメモリアドレスのタグ(501)の値とエントリ内に格納されたタグの値(605および606)を比較器601で比較する。タグの値が等しいエントリ内にはメモリロケーション内容の写しがキャッシュエントリに格納されていると判断する。すなわち、メモリアドレスのタグ値(501)とキャッシュエントリのタグ値(605または606)が等しければキャッシュヒットとなる。この場合メモリアクセスはキャッシュアクセスに置き換えられ、高速アクセスが可能となる。タグの値が等しいエントリがない場合、キャッシュ内にメモリロケーションの写しが格納されていないと判断し、キャッシュアクセスの代わりに低速なメモリアクセスが必要となる。

【0015】キャッシュ内にメモリロケーションの写しがない場合はさらに、その写しをキャッシュに格納する。これは以下のように行う。タグの値の等しいエントリがなかった場合は、2つのエントリの制御ビットのemptyビットを調べる。少なくともどちらか一方のemptyビットが真であれば、そのキャッシュエントリが空きであるので、メモリから読み出したデータをそこ

に格納する。いずれのエントリのemptyビットが偽であれば、2つのうち一方をキャッシュから追い出す。このときどちらを追い出すかの判断にreuseビットとrecentビットを用いる。これを表で示したのが図1であり、以下これを説明する。図1は2つのエントリ1(603)とエントリ2(604)のうち、どちらを追い出しエントリとするか、選択する様子を示す図である。エントリ1(603)はセット1に、エントリ2(604)はセット2に属する。2つのセットのエントリの制御ビットの中のreuseビットとrecentビットを、追い出しエントリ選択手段101の入力とし、追い出すエントリを選択する。選択手段101は上記の4つのビットを入力とし、0か1かを出力する。0はエントリ1を1はエントリ2を選択することを意味する。入力と出力の関係を示したのが図1(b)の表102である。この表の意味するところを説明すると次のようになる。2セットのうち一方だけにreuseビットが立っている(真になっている)場合は、そうでない(偽になっている)方のセットのエントリを選択する。両セットのreuseビットとも真、または両セットとも偽になっている場合は、2セットの内の古い(recentビットが偽)エントリを選択する(recentビットは初期状態では両セットとも偽となっているが、そのアクセスがあれば必ず一方のセットが真、他方が偽となっていて、両方が真となることはない)。recentビットが両方とも偽になっている場合はどちらを選択してもよい(0/1により表示)。追い出すエントリが決まったら、そのエントリ内容が変更されているかどうか(dirtyビットが真かどうか)調べ、そうならばこのエントリの内容をメモリに書き込む。変更されていないければ(dirtyビットが偽)何もしない。古いエントリの追い出しが完了すれば、メモリロケーション内のデータを読み出しその値を、追い出したキャッシュエントリに格納し、そのエントリのrecentビットを真にし、追い出されなかった方のエントリのrecentビットを偽とする。

【0016】図7は本発明で適用するロード命令と、それがキャッシュエントリのreuseビットに設定される様子を示した図である。ロード命令の命令語702にはREUSE指定があるかどうかを示すビット(704)を含む。このビットが真であれば、当該ロード命令でアクセスしたデータを含むキャッシュエントリ(306)のreuseビット(401)は真となる。

【0017】次に図8のプログラムを例にとって、従来のキャッシュシステムと本発明のキャッシュシステムの振舞いの違いを説明するが、その前に従来のキャッシュラインの制御ビットと追い出しエントリの選択方法について簡単に説明する。なお、図8において、intは整数(s)の宣言を意味し、t+=2は2を加えることを意味する。従来のキャッシュシステムにおけるキャッシ

ュラインの制御ビットはemptyビット、recentビット、およびdirtyビットの3つのビットから構成されている。そして、キャッシュラインが競合したときは、recentビットが偽のエントリ、すなわち最後にアクセスされたのがより古い方のエントリをキャッシュから追い出すようにしている。図8のプログラムの2重ループにおいて、外側ループ(802)の1回目のイタレーション(繰返し(単位)、例えば、99回のループがあるとき、その各回をイタレーションという)が完了したとする。この時点で、配列a、bのアクセスした部分は次の通りである。

a[0..1][1..1023], b[1..1023]

ここで例えばb[1..1023]はb[1], b[2], ..., b[1023]を表す。このうち、配列aは外側ループのイタレーションの進行に伴いアクセス位置が変化していく。一方配列b(外側ループについては)はアクセス位置が変化しない(というが配列b全体であるが、その全容量はサイズの的にキャッシュに収まる)。したがって配列b[1..1023]が長い間キャッシュに保持され、外側ループの個別のイタレーションで再利用できるなら高速化の面で都合がよい。しかし、2ウェイセットアソシアティブキャッシュの場合には配列bがずっとキャッシュに保持されるとは限らない。というのは配列bの要素b[j]は配列aの要素a[i][j]、要素a[i+1][j]とキャッシュラインを競合するからである。それを以下説明する。なおここで、2次元配列は2次元目(jの方)が速く変化するようにメモリ上に配置されるとする。図5で示したように、あるアドレスのデータが入るべきキャッシュライン(インデクス)はその中位7ビットによって決まる。従って2つの配列要素があったときそのアドレスの中位7ビットが等しければそれらは同じラインに入る(キャッシュラインの競合)。ただし2ウェイセットアソシアティブであれば1つのインデクスに2つのエントリが対応する。b[j]のアドレスは「配列bの先頭アドレス(b[0]のアドレスのこと)+j*4」、a[i][j]のアドレスは「配列aの先頭アドレス(a[0][0]のアドレスのこと)+i*4096+j*4」である。よってa[i][j]とa[i+1][j]のアドレスはちょうど4096だけ異なり、この2つは明らかに同じキャッシュラインに入る。また、aの先頭アドレスとbの先頭アドレスの差が4096の倍数であれば(aの大きさは4096バイトなので、これは成り立つ可能性が高い)a[i][j]とb[j]もやはり同じキャッシュラインに入る。故に、b[j], a[i][j], a[i+1][j]はキャッシュラインを競合する。

【0018】図9は、図8のプログラムを機械語イメージに近付けたプログラムである。機械語イメージに近付

けたというのは、いま注目している配列a、bへのアクセスを明示的にload()により示しているということである。ここで、r0、r1、r2はレジスタを指す。ループ内の配列要素アクセスはa[i][j]、a[i+1][j]、b[j]の順に行われる。内側ループのイタレーションを固定(jの値が固定のJ)し、外側ループのイタレーションを順次に増大させて以下従来のキャッシュの振舞いを考える。さて、外側ループの初回のイタレーション(iの値は0)において、3つの配列要素参照a[0][J]、a[1][J]、b[J]はキャッシュラインを競合し、そして置換方式がLRUであるので、3つのうちa[1][J]とb[J]がキャッシュに残る。このうちb[J]のrecentビットが真となる。外側ループの次のイタレーション(iの値は2)では、最初a[2][J]がアクセスされる。この時点で対応するキャッシュエントリにはa[1][J]とb[J]が残っているが、a[1][J]の方のrecentビットが偽であるので、a[1][J]がキャッシュから追い出され、a[2][J]がキャッシュにはいる。その際追い出されなかった方のb[J]のrecentビットが偽になり、新しくキャッシュに入ったa[2][J]のrecentビットが真になる。次にa[3][J]がアクセスされ、これに伴いb[J]がキャッシュから追い出され、a[3][J]がキャッシュにはいる。最後にb[J]がアクセスされ、a[2][J]がキャッシュから追い出されb[J]がキャッシュにはいる。外側ループイタレーションの3回目以降は2回目以降と同様になる。ここで注目すべきはb[J]が一旦追い出され、それから新たにフェッチされている点である。つまり外側ループイタレーションの任意の回(iが任意の値、jが任意の値J)で、b[J]のメモリからのフェッチが発生している点である。jの値は任意であったのでこれは配列bの全体がキャッシュに全く残っていなかったことを意味する。以上、従来のキャッシュシステムではキャッシュをまったく有効に活用できないことを示した。

【0019】次に、本発明のキャッシュシステムで上記のプログラムがキャッシュを有効利用できることを示す。図8を本発明のシステムの機械語イメージに近付けたプログラムを図10に示す。このプログラムではメモリアccessをload()で明示指定し、その際REUSEかどうかを指定してある。さて、本発明のキャッシュで先程と同様に図10のプログラムの実行を追ってみる。内側ループのイタレーションを固定(jの値が固定のJ)し、外側ループのイタレーションを順次に増大させて以下考える。外側ループの初回のイタレーション(iの値は0)において、3つの配列参照a[0][J]、a[1][J]、b[J]はキャッシュラインを競合し、そして置換方式がLRUであるので、3つのうちa[1][J]とb[J]がキャッシュに残る。そ

してb〔J〕はrecentビットが真となる(a〔1〕〔J〕はrecentビットが偽)。さらにb〔J〕のアクセスはREUSE指定であり(1006)そのreuseビットも真となる。外側ループの次のイタレーション(iの値は2)でも3つの配列参照a〔2〕〔J〕、a〔3〕〔J〕、b〔J〕がある。最初a〔2〕〔J〕がアクセスされることに伴いa〔1〕〔J〕がキャッシュから追い出され(reuseビットもrecentビットも偽のため)、a〔2〕〔J〕がキャッシュに入る。次にa〔3〕〔J〕がアクセスされ、これに伴いa〔2〕〔J〕とb〔J〕のどちらかがキャッシュから追い出されることになる。a〔2〕〔J〕はreuseビットが偽、recentビットが真であり、一方b〔J〕はreuseビットが真、recentビットが偽である。従って図1で示した通りa〔2〕〔J〕が追い出されるエントリとして選ばれ、b〔J〕が残る。そしてa〔3〕〔J〕がキャッシュに入る。最後にb〔J〕がアクセスされるが、この場合b〔J〕がすでにキャッシュにあり(キャッシュヒット)、メモリからロードする必要はない。もちろんb〔J〕はrecentかつreuseとなる。外側ループイタレーションの3回目以降は2回目とはほぼ同様になるが、念のために3回目も示しておく。外側ループの3回目のイタレーション(iの値は4)では、a〔4〕〔J〕、a〔5〕〔J〕、b〔J〕の3つが参照される。最初a〔4〕〔J〕がアクセスされこれに伴いa〔3〕〔J〕がキャッシュから追い出されa〔4〕〔J〕がキャッシュに入る。次にa〔5〕〔J〕がアクセスされこれに伴いa〔4〕〔J〕とb〔J〕のどちらかがキャッシュから追い出されることになる。a〔4〕〔J〕はrecent、b〔J〕はreuseであり、したがってa〔4〕〔J〕が追い出される。そしてa〔5〕〔J〕がキャッシュに入る。最後にb〔J〕がアクセスされこの場合b〔J〕がすでにキャッシュにあり、キャッシュヒットになる。ここで注目すべきはb〔J〕が一旦キャッシュにフェッチされると以降は一度も追い出されていない点である。つまり外側ループイタレーションの初回で配列b全体がメモリからキャッシュにフェッチされると、以降フェッチが全く発生していない点である。言い換えると、配列bへのアクセスは時間的局所性をフルに引き出せたということで、これが本発明のキャッシュの利点である。

【0020】

【発明の効果】本発明によれば、キャッシュの各エントリが、そのエントリのデータをなるべくキャッシュから追い出さないようにすることを示すreuse情報を持つので、キャッシュ置換時、どのエントリをキャッシュから追い出すかを決定する際、この情報を利用できる。すなわち、その情報を持つエントリを優先的にキャッシュ

に残すようにできる。また、メモリデータの参照とreuse情報の値の設定をする命令の実行に応じて、エントリにデータを格納すると共にreuse情報の値が設定されるので、ユーザまたはコンパイラが特定のデータをなるべくキャッシュに残すように指示することが可能となる。そして、キャッシュミスが少なくすることが可能となる。

【図面の簡単な説明】

【図1】本発明の一実施例の追い出しエントリの選択に係る構成および選択を説明するテーブルを示す図である。

【図2】キャッシュを含む情報処理装置の構成を示す図である。

【図3】本発明の一実施例のキャッシュの構成を示す図である。

【図4】本発明の一実施例のキャッシュの1ラインの構成を示す図である。

【図5】メモリアドレスの構成を示す図である。

【図6】キャッシュアクセスの仕組みを説明するための図である。

【図7】本発明で適用されるロード命令および該命令のキャッシュに対する作用を示す図である。

【図8】キャッシュの動作を説明するためのプログラムを示す図である。

【図9】従来のキャッシュの動作を説明するために図8のプログラムを機械語イメージに近づけたプログラムを示す図である。

【図10】本発明のキャッシュの動作を説明するために図8のプログラムを機械語イメージに近づけたプログラムを示す図である。

【符号の説明】

101 追い出しエントリ選択手段

201 CPU

202 キャッシュ

203 バス

204 メモリ

301 セット1

302 セット2

303 制御ビット

304 タグ

305 データ

306 ライン

307 インデックス

401 emptyビット

402 recentビット

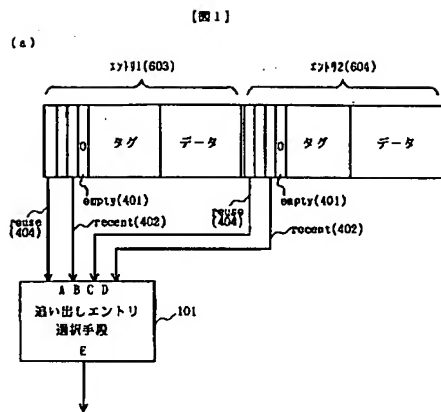
403 dirtyビット

404 reuseビット

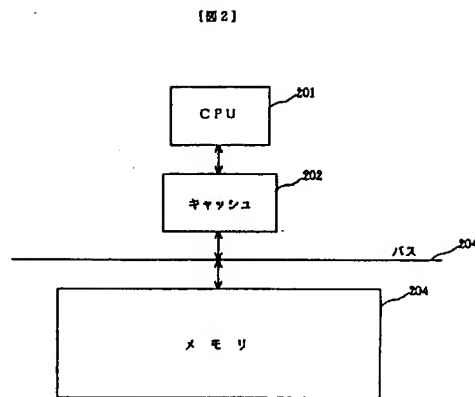
603 エントリ1

604 エントリ2

【図1】



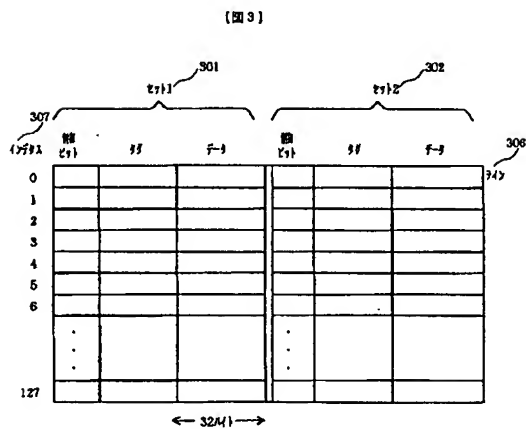
【図2】



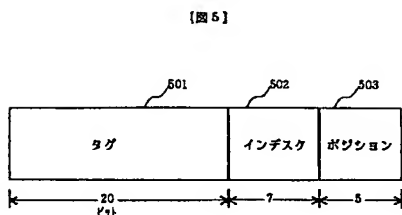
(b)

A	B	C	D	E
0	0	0	0	0/1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	1	0	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0/1
1	0	1	1	0
1	1	0	0	1
1	1	1	0	1

【図3】

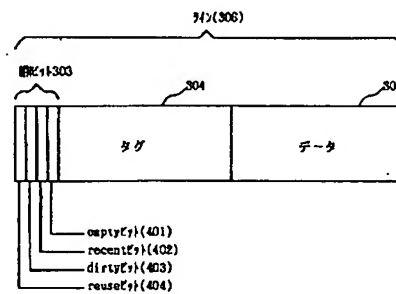


【図5】

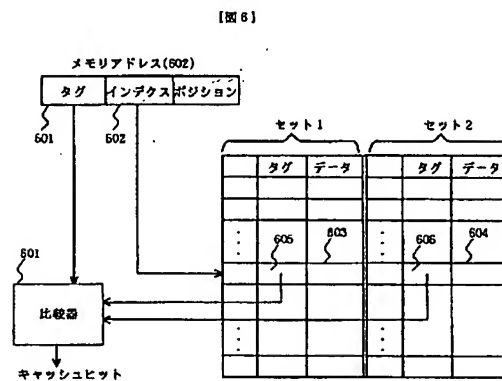


【図4】

【図4】

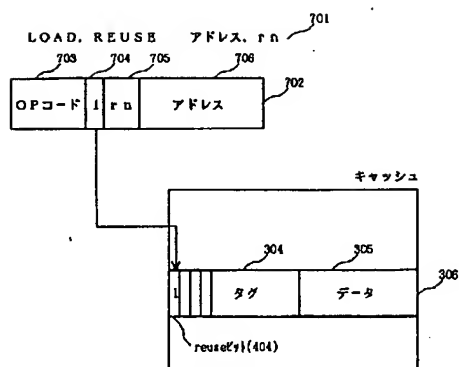


【図6】



【図7】

【図7】



【図8】

【図8】

```

int s,a[1024][1024],b[1024];
for (i=0; i<1024; i+=2)
  for (j=0; j<1024; j++)
    s=s+a[i][j]+a[i+1][j]+b[j];

```

801
802
803
804

【図10】

【図10】

【図9】

【図9】

```

int s,a[1024][1024],b[1024];
for (i=0; i<1024; i+=2)
  for (j=0; j<1024; j++){
    load(r0, &a[i][j]);
    load(r1, &a[i+1][j]);
    load(r2, &b[j]);
    s=s+r0+r1+r2;
  }

```

901
902
903
904
905
906
907
908

```

int s,a[1024][1024],b[1024];
for (i=0; i<1024; i+=2)
  for (j=0; j<1024; j++){
    load(r0, &a[i][j], NO_REUSE);
    load(r1, &a[i+1][j], NO_REUSE);
    load(r2, &b[j], REUSE);
    s=s+r0+r1+r2;
  }

```

1001
1002
1003
1004
1005
1006
1007
1008